Portland State University

PDXScholar

Computer Science Faculty Publications and Presentations

Computer Science

10-1997

A Player for Adaptive MPEG Video Streaming Over The Internet

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

Rainer Koster
Oregon Graduate Institute of Science & Technology

Shanwei Cen
Oregon Graduate Institute of Science & Technology

Crispin Cowan
Oregon Graduate Institute of Science & Technology

David Maier
Oregon Graduate Institute of Science & Technology

Follow this and additional works at: https://pdxscholar.library.pdx.edu/compsci_fac

 Part of the Computer and Systems Architecture Commons, and the Digital Communications and Networking Commons

See next page for additional authors

Let us know how access to this document benefits you.

Citation Details

Jonathan Walpole ; Rainer Koster ; Shanwei Cen ; Crispin Cowan ; David Maier ; Dylan McNamee ; Calton Pu ; David C. Steere and Liujin Yu, "Player for adaptive MPEG video streaming over the Internet", Proc. SPIE 3240, 26th AIPR Workshop: Exploiting New Image Sources and Sensors, 270 (March 1, 1998); doi:10.1117/12.300064; http://dx.doi.org/10.1117/12.300064. Copyright 1997 Society of Photo-Optical Instrumentation Engineers. One print or electronic copy may be made for personal use only. Systematic reproduction and distribution, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper are prohibited.

This Conference Proceeding is brought to you for free and open access. It has been accepted for inclusion in Computer Science Faculty Publications and Presentations by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Authors

Jonathan Walpole, Rainer Koster, Shanwei Cen, Crispin Cowan, David Maier, Dylan McNamee, Calton Pu, David Steere, and Liujin Yu

This conference proceeding is available at PDXScholar: https://pdxscholar.library.pdx.edu/compsci_fac/56

A player for adaptive MPEG video streaming over the Internet

Jonathan Walpole, Rainer Koster, Shanwei Cen, Crispin Cowan, David Maier, Dylan McNamee, Calton Pu, David Steere and Liujin Yu

Department of Computer Science and Engineering
Oregon Graduate Institute of Science and Technology
P.O. Box 91000, Portland, Oregon 97006

ABSTRACT

This paper describes the design and implementation of a real-time, streaming, Internet video and audio player. The player has a number of advanced features including dynamic adaptation to changes in available bandwidth, latency and latency variation; a multi-dimensional media scaling capability driven by user-specified quality of service (QoS) requirements; and support for complex content comprising multiple synchronized video and audio streams. The player was developed as part of the QUASAR† project at Oregon Graduate Institute, is freely available, and serves as a testbed for research in adaptive resource management and QoS control.

Keywords: Internet, video, real-time, MPEG, adaptive, feedback, quality of service

1. INTRODUCTION

Digital multimedia systems are becoming ubiquitous, with nearly all computer platforms offering support for real-time datatypes such as audio and video. There has also been a rapid proliferation of communications networks, giving multimedia computing the potential to augment, or even replace, traditional broadcast and print media with more interactive and personalized information services. Compressed digital video[1] and the Internet are technologies at the heart of this revolution. This paper addresses the design and implementation of a player for interactively streaming video and audio data, in real-time, across the Internet.

Although the promise of ubiquitous multimedia computing and networking is exciting, today's real-time multimedia applications tend to be resource-hungry and inflexible in the presence of contention for resources. For example, on shared computing platforms, when CPU, network or disk bandwidth become short, multimedia applications are often unable to maintain the real-time playout of video and audio data. One proposed solution to this problem is for underlying systems software to support resource reservations in order to provide QoS guarantees to applications[2,3]. To solve the problems of inflexible multimedia applications, such reservations must be end-to-end, covering all system components, including network and end-host resources.

Although resource reservation has received significant attention within standards bodies such as the IETF[4,5], it is likely to take a long time to achieve wide-spread deployment of such end-to-end, system-level solutions. In the meantime, multimedia systems researchers are exploring alternative solutions[6]. One such alternative is to build *adaptive* multimedia applications that attempt to preserve presentation quality in one dimension, by allowing quality to vary in other dimensions when resources become scarce[7,8,9]. For example, when available network bandwidth becomes scarce, an adaptive video player could reduce presentation quality in the frame rate dimension (i.e., temporal resolution) in order to preserve real-time playout at a given spatial resolution.

The appropriate way to trade quality among various quality dimensions is both user and task-dependent. For example, a user trying to identify license plates from a surveillance video would probably attach a higher value to spatial resolution than frame rate. In contrast, a user watching a basketball game may attach a higher value to frame rate than spatial resolution. To support this variety of preferences, adaptive multimedia systems should allow higher layers to specify their QoS requirements[9,10,11].

In this paper, we describe the architecture of an adaptive Internet-based video and audio player. The player supports standard VCR facilities, such as play, fast forward, rewind, pause, etc., for real-time streaming of stored MPEG-com-

†. QUASAR is an abbreviation for QUAlity Specification and Adaptive Resource management.

SPIE Vol. 3240 • 0277-786X/98/$10.00

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

pressed video. It also takes advantage of the inherent flexibility of a software solution to go beyond a simple network TV, and offer several advanced features, including *complex content* comprised of multiple synchronized video and audio streams, simultaneous synchronized access to *distributed servers, prefetching* to hide remote access latency, and *adaptive QoS control* in several quality dimensions. These features allow the player to support advanced applications and make it robust in heterogeneous environments that have a high variability in the capacity of their communication and computation resources.

The paper is organized as follows. Section 2 outlines the requirements of several advanced multimedia applications that motivate the player's novel features. Section 3 introduces the QoS model and adaptation mechanisms used in the player. Section 4 presents a detailed description of the player's architecture, and Section 5 discusses its performance. Finally, Section 6 concludes and discusses future work.

2. REQUIREMENTS

This section illustrates three advanced multimedia streaming applications with requirements that are not addressed by current multimedia presentation tools. Such applications need support for complex presentations, composed of several synchronized video or audio clips, and control over several quality dimensions

2.1 Electronic News Gathering

Television production is moving towards the all-digital studio, in which a single, high-bandwidth data network replaces the multiple, dedicated analog data and control channels of conventional studios. Electronic news gathering is one application of the digital studio[12]. In a local television studio, several editors may simultaneously produce broadcast segments for a daily news show, combining video footage from station archives with network newsfeeds and recently taken local coverage, along with graphics and voice overs. Simultaneously, video may be captured to storage and previously edited segments played out for broadcast. In selecting video clips for inclusion, the news editor might play back video in forward or reverse modes at up to 100 times the normal speed. However, in choosing exactly where to start and end a clip, the editor might want to slow down playback, or even step through frame by frame. In determining when to cut between two camera angles – such as of a public official and a questioner in the audience – the editor will want to see the video from the two angles played back simultaneously, synchronized with each other and with the corresponding audio. Once the segment is composed from various clips, the news editor will want to review it at normal speed and at near broadcast quality.

This editing scenario highlights several demands on multimedia streaming systems. First, there can be multiple competing users of such systems, and it must be possible to share resources and prioritize their usage. For example, playback of a segment for actual broadcast should take priority over editing tasks, but two editors should share resources roughly equally. Second, user actions, such as changing play speed and direction, have potentially complex impact on presentation characteristics. For example, speeding up playback to 100 times normal speed shouldn't cause the system to attempt to display 3000 frames per second. The system should select a subsequence of frames to display that provides equivalent quality to normal-speed playback. Third, the level of playback quality is not constant – rough cuts of a news story can be displayed with lower quality than that used during final review of the story or during broadcast. Different quality dimensions can also be seen in this example – for example, frame rate and synchronization of streams.

The editing scenario also has the requirement to deliver multiple streams at once, potentially from different servers. This requirement arises when editing a news segment that uses both archival and current video footage resident on different servers. While it is possible to copy all the clips, in advance, onto a single server, that approach is often impractical. For example, during the early iterations of the editing process, the actual selection and length of clips changes frequently, and hence pre-copying clips then can be prohibitive.

2.2 Sports Video on Demand

Staehli[11] describes a sports video-on-demand application, using professional basketball as an example. Sports events are often recorded using several cameras and microphones, providing raw footage composed of multiple viewpoints and sound tracks. Interesting actions, such as plays or fouls, may only be visible from some camera angles, and hence access to the entire raw footage is valuable to support activities such as customizable, interactive replays. Using current multimedia presentation technology, however, viewers only have access to a single edited presentation, and there is minimal support for user interaction.

A sports video on demand system that provides access to all the video and audio recordings of an event would create a new, more interactive way of viewing sports. Users could have a small window for each camera and dynamically select

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

what to view on the main screen. Replays could be selected from specific camera angles, and viewed in slow-motion, while the game continued in additional windows. Commentators and statistics could be provided, as required, in additional windows. To summarize, the user of such a sports video on demand system could have access to all of the information that is currently available only to TV editors. In such an application, the editor may still play a role, perhaps defining a default view of the sporting event that can be further customized by the viewer if desired.

If such complex and interactive presentation applications are to be extended into the home, across networks such as the Internet, adaptive management of scarce resources such as network bandwidth will become critical. In the example above, small windows providing additional views may be assigned a lower quality than the main window, hence reducing their resource consumption. For displaying action in sports events, a high frame rate is often needed. This requirement may be supported in the presence of resource contention by lowering the spatial resolution of the video. The statistics board, on the other hand, would be displayed with high spatial resolution to make it readable, but with very low frame rate. A picture of a commentator may not be important, but may be nice to have when resources suffice. In any case, commentator audio would likely take precedence over commentator video.

2.3 Intelligence Analysis

Consider an intelligence analyst assembling a multimedia presentation, to be communicated with other analysts via an internetwork, to back up an assessment of the health of a foreign leader. Part of the content might be composed of video segments taken from television broadcasts, photos from news agencies, and audio clips from radio addresses. Different parts of the presentation require emphasizing different aspects of quality for optimal interpretation. One part of the presentation is a sequence of video clips and color images arranged by date, in order to illustrate changes in complexion and weight. Here the analyst wants to emphasize individual image quality, both in color fidelity and spatial resolution. Another part of the presentation shows recent video clips of the leader walking and greeting visitors, to show stiffness and slowness of movement. The analyst is less concerned with image detail in this segment, but wants to specify that the timing of the presentation be very accurate, so as not to introduce artificial jerkiness. Next comes a clip of a televised interview, where the analyst deems audio fidelity most important, to demonstrate slurred speech. A fourth part of the presentation shows two video segments side by side, to demonstrate that what is being offered as recent coverage of the leader in his office strongly resembles coverage shown two months ago, but shot from a different angle. Here the most important aspect of quality to the analyst is having the two segments stay precisely synchronized.

Under optimal conditions, an analyst viewing the presentation remotely may be able to play it back with near-perfect image quality, timing, audio fidelity and synchronization throughout. More likely, however, is that the presentation must be shown to several clients under a variety of circumstances with different computers, different network connections and competing processing going on. In these cases, the QoS specifications that the authoring analyst has attached to the various parts of the presentation should be available to influence resource management decisions. Ideally, the most important aspects of the content should be given the resources they need for accurate rendering. Similarly, the QoS requirements of the viewing analysts should also be available to influence resource management, so they get the quality needed to support their task.

2.4 Summary of requirements

The examples above have shown a variety of requirements for advanced multimedia streaming applications. First, users should be able to combine data to form *complex presentations*. Operators such as concatenation, synchronization (concurrent presentation), and clipping (selecting particular parts) of streams need to be supported. Modifications of clips such as changing the audio gain, scaling a video image, or including a stream as slow motion are also useful. Second, the content of a presentation can be *distributed*, since material from different remote servers may be included by reference in a presentation. During play-back, data must be retrieved from those locations and displayed in real-time. Third, such applications need to provide *adaptive QoS control*. For video, important QoS dimensions include spatial resolution, frame rate, color accuracy, and temporal jitter. If several streams are combined, synchronization is another important quality dimension. Moreover, different parts of a presentation may be assigned different priorities. The user should be able to trade reduced quality in some dimensions for increased quality in others.

3. BACKGROUND

3.1 QoS model

Most existing multimedia presentation systems tend either to not manage quality explicitly at all, or try to control only a single quality dimension, such as frame rate. In addition, quality is often tightly coupled to presentation content and view in inflexible ways. Examples of this excessively tight coupling include binding the frame rate and resolution of a

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

video presentation to the capture frame rate and resolution of the video source. Another example is the use of playback speed to determine frame rate. In order to understand how to build multimedia presentation systems with more flexible and independent quality control, either driven by the user directly or via feedback mechanisms, we have developed a model in which the concepts of *content, view* and *quality* of a presentation are separated.

In our model, the *content* of a presentation is specified as a hierarchical structure with single-medium sources at the leaves, and the internal nodes representing various transforms and compositions on that base content, such as clipping, scaling and concatenation. For example, a content definition might specify 3 seconds of video source A, followed immediately by 4 seconds of video source B, both proceeding in parallel with audio clip C.

The *view* definition of a presentation specifies an idealized mapping of the content into a display space. In general, the content of a presentation is determined by the author of the presentation, while the view is selected by each consumer of that presentation. An example view might indicate that the content be presented in a certain 6-by-8 cm rectangle on the screen, that the presentation should proceed at half normal speed, and that color content should be mapped to grayscale values. The separation of content and view is essential when the author of the presentation can foresee neither all uses to which it will be put nor all environments where it will be played.

The final component of our model, *quality*, describes the allowable divergence between the ideal presentation (as determined by content and view) and the actual presentation delivered to the customer.

This Content-View-Quality model was formally defined in the Z specification language[10,11]. The quality aspect of the formalization introduces an error model with multiple error components, and recognizes that the divergence of the actual presentation from the ideal presentation can usually be interpreted in more than one way relative to those components. For example, divergence or inaccuracy in an audio stream can always be explained as an amplitude error, but might sometimes be explicable as a phase-shift error. Thus, the formal model provides a weighting of different error components (indicating severity of that kind of error) and takes the "optimistic" interpretation of the divergence. An overall error measure then results from combining the weighted errors in each component.

We have used our model to construct a multimedia player that tries to determine a presentation plan that guarantees a given error bound on quality for a specified content and view[11]. However, the system described here does not try to guarantee a hard error bound on quality. Rather, we use quality preferences and bounds supplied by the user to choose which aspects of quality to degrade when our feedback mechanism (illustrated in Figure 1) detects that resources are insufficient, and also to mark the point at which quality is adequate, i.e., where no further resources should be consumed in an attempt to improve it.

In the system described here, a presentation can be one or more simultaneous video streams along with zero or one audio stream. Each individual stream is the concatenation of clips from stored media sources, possibly with gaps. The sources can be distributed on multiple servers. The view parameters that the user can control are location and size of the display of each video stream, and the playback speed of the entire presentation. The location and size aspects are modified simply by dragging and resizing the appropriate display window, while playback speed is controlled by a slider and VCR-style controls. The quality aspects under explicit user control are framerate and spatial resolution. As resources can not be guaranteed in the Internet environment, the user actually selects maximum values for each, plus the relative importance of each aspect, for when they have to be scaled back because of limited resources. There are two other aspects of quality—synchronization and temporal jitter—that are not under explicit user control, but which our feedback mechanism attempts to optimize.

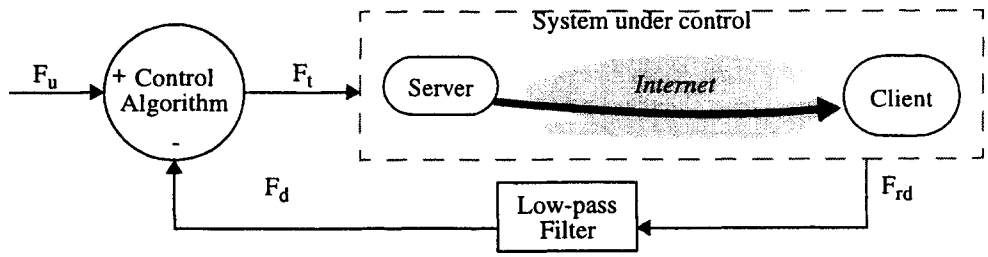
While we do not require users of our player to master the subtle distinctions between view and quality aspects, we do believe our architecture preserves their independence. For example, enlarging the display window for a video stream does not force it to a higher resolution. Rather, the image is scaled with the same number of pixels "fattened" to expand into the available space.

3.2 Software feedback-based adaptation

Software feedback is a technique that uses feedback mechanisms, derived from control theory, similar to those in hardware feedback systems. A feedback system monitors the output and internal state of a system under control, compares them against a goal specification, and feeds the difference back to adjust the behavior of the system. One beneficial property of feedback mechanisms is that they can control complex dynamic systems adaptively and efficiently even though they have very little knowledge of the target system's internal structure.

The QUASAR player contains a number of end-to-end feedback systems to allow it to adapt its network bandwidth consumption, QoS, and client server synchronization. All of these feedback mechanisms were constructed using the SWiFT software feedback toolkit[13] which allows selection of feedback components from a component library, and supports tuning using simulation and run-time instrumentation tools. Sections 4.2 and 4.3 discuss these feedback mechanisms in more detail.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms



F_{rd} - raw display frame rate F_u - user-specified frame-rate

F_d - filtered display frame rate F_t - target frame rate

Figure 1. QoS control feedback

4. ARCHITECTURE

The player has a multi-client, multi-server architecture. Video (MPEG-1) and audio (MPEG 1 or μ-law) data streams are transmitted across the Internet from one or more servers to one or more clients. An individual server may be servicing multiple clients simultaneously, and a single client may be receiving data from several servers simultaneously. Servers are able to serve several clients at a time by forking a video server (VS) or audio server (AS) process per stream. These processes operate independently of each other. The player architecture is illustrated in Figure 2.

Between the client and each server, there is a TCP-based control connection and a UDP-based data connection. The client sends commands such as `play`, `rewind`, or `stop` to the servers. When playing, VS and AS periodically read data from disk, send a packet over the UDP connection, and block for a certain length of time. While playing, the client can request a server to change quality or to adjust the timing. Since audio requires little bandwidth compared to video, and gaps in an audio stream are easily perceived by the user, the player tries to resend lost audio packets once. No attempt is made to resend lost video packets.

VS needs to locate frames in the MPEG video file in order to select clips from a file, position a presentation to a particular point, and switch files to support resolution change requests (Section 4.2). Each server maintains a frame index for each video file that includes frame sizes, frame position in the file, and information about MPEG headers and groups of pictures. To generate this index, the MPEG stream is parsed the first time it is opened. Because this parsing can take several seconds, the index is saved on disk to speed up future accesses.

The client, illustrated in Figure 2, consists of several processes† and buffers. In the client, there are three major components: control structures, a video pipeline, and an audio pipeline. (The figure assumes video data is compressed and audio is uncompressed.) The video buffering (VB) and audio buffering (AB) processes receive video and audio packets and put them into buffers. The buffering is necessary to remove network jitter. VB also reassembles video frames that have been chopped into several network packets. The video decoder process (VD) does the MPEG decoding and dithering into images ready for display. These images are put into another buffer (B2). The buffers are located in shared memory, allowing data to be passed to other processes quickly. Access to buffers is synchronized using semaphores.

The video play (VP) process and the user interface (UI) process maintain connections to the local display server (such as an X-window) on the client. VP displays the video images. The control process CTR handles the timing and releases the video frames from B2 to VP to be displayed. In addition, it transfers the corresponding audio samples to the audio device. The UI has controls for view, including VCR-style buttons, and quality. UI manages the user interface and sends user events to CTR via sockets. Global status variables are stored in shared memory and can be accessed by all processes.

Any stage in the video pipeline can drop frames if resources are insufficient. For example, frames can be dropped if packets are lost in the network due to congestion, if B1 overflows, if the decoder is too slow (insufficient CPU cycles), or

†. Processes rather than threads have been chosen for better portability, however a multi-threaded version is also under development.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

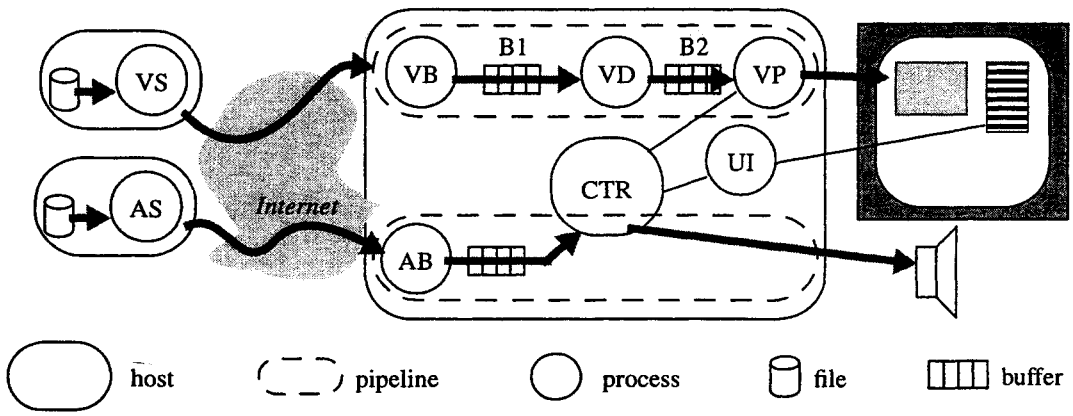


Figure 2. Player architecture

if the frame is too late when it eventually arrives in B2.

Particularly at the decoder stage, MPEG's inter-frame dependencies have to be taken into account. For instance, if an I-frame that is 10 msec late was dropped by VD, all subsequent frames referring to it cannot not be decoded, resulting in a gap of about 400 msec for a common frame pattern. Hence, an I-frame is dropped only if it is more than 400 msec late. While the I-frame itself will not be displayed if it is late, it is still useful for decoding subsequent P- and B-frames.

4.1 QoS control

Our QoS model keeps content, view, and quality specifications independent of each other. This goal requires the architecture to support changes in quality in any QoS dimension without affecting the view, and vice versa.

Our current implementation supports quality and view control in the temporal and spatial dimensions. For example, view is defined by play speed and image size parameters, whereas quality is defined by frame rate and (spatial) resolution. Section 4.4 discusses support for content specification.

In general, we expect quality and view parameters to be either manipulated directly by users, or adjusted automatically via feedback-based adaptive resource management mechanisms. In the player, play speed (the ratio of playout speed to normal speed) and is specified manually by a slider, and the image size is specified by resizing the output window using the window manager. Frame rate (the number of frames displayed per second) may be controlled automatically or manually. Similarly, resolution may be chosen from a menu or adjusted automatically. Finally, the user may choose to link the image size and resolution such that the image size changes automatically when the resolution changes.

Play speed and frame rate control the mapping of video frame numbers to real time using the client's system clock. Figure 3 shows a naive mapping for some example play speeds and frame rates, where the source is encoded at 30 frames per second. However, this naive mapping has to be modified in practice to take account of MPEG's inter-frame dependencies. In MPEG, there are three types of frames: I, P, and B. The groups of pictures (GOP) structure defines a specific frame ordering pattern such as IBBPBBPBBPBB. B-frames and P-frames can only be decoded after the decoding of the I and P-frames they depend on. Only I-frames are decodable on their own. This restriction requires a sophisticated way of dropping frames in order to achieve a specified frame rate. Figure 4 shows how send patterns are generated for various framerates during playback at normal speed. All B-frames are dropped before any P-frames, and all P-frames are dropped before any I-frames. Based on this pattern, the video server selects the frames to be sent to the client.

To adapt quality in accordance with the preferences of the user, we have chosen spatial resolution as a second variable quality dimension. Since MPEG-1 does not provide several quality levels for one source file, we use separate files for different resolutions. Storing the same content several times at the server is a very simple way of providing scaling without support by the compression algorithm and is possible with reasonable overhead. Providing files with four resolution steps requires only about twice as much disk space as having the highest resolution only.

A configuration file at the server side is used to specify that a set of files are actually the same video at different resolutions. This file, plus the index files, form a logical index. The video server checks that the lengths and frame patterns of all files are the same. When changing the resolution, the server simply switches to another source file. To locate the position of a particular frame in a file, index tables are used. They are generated once and stored on disk for future accesses. These tables allow random access to frames while searching for a position in the video clip, fast forwarding, and skipping frames while playing. When a video with multiple resolutions is initialized, the indices of all files are loaded into main

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

Real time (msec)		0			55			110			165			220			275			330
Play Speed	Frame Rate																			
100%	30	1		2		3		4		5		6		7		8		9		10		11
100%	15	1				3				5				7				9				11
150%	45	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			
150%	30	1	2		4	5		7	8		10	11		13	14		16			
50%	15	1			2			3			4			5			6			

Figure 3. Frames displayed for various combinations of play speed and frame rate

memory. This approach allows quick switching among different resolutions by simply using another file and another index for locating a particular frame. However, such switches must occur on I-frame boundaries.

The player allows the size of the video output to be different from the resolution in the source file, i.e., the view of the image can be scaled. To implement variable image size, a scaling step has been added to VD between the decoding and dithering of video frames. Additionally, the dithering algorithms, buffer B2, and VP must be able to deal with dynamically changing image sizes. Currently, scaling and dithering are done in two passes over an uncompressed frame. A possible performance improvement is to combine these operations into a one-pass algorithm.

Ideally, resource consumption would depend only on the presentation quality provided and would be independent of changes in view. While this situation holds true for play speed and frame rate in our architecture, it is not the case for resolution and image size. While resolution, a quality parameter, controls resource usage for transmitting and decoding frames, CPU consumption for dithering depends on the image size, which is a view parameter.

4.2 Adaptation

As mentioned above, any overloaded pipeline stage can drop video frames. While the VD stage takes MPEG frame dependencies into account before choosing a frame to drop, the network and VB drop frames at random, a characteristic that can cause high jitter in the presentation. When a frame is dropped, all resources used earlier in the pipeline for processing the dropped data are wasted. For instance, a frame that is dropped by the control process, because it arrives too late in B2, has already consumed disk bandwidth, memory, and CPU on the server, and bandwidth on the network, as well as client memory and CPU cycles for decoding. To avoid wasting resources in this manner, the QUASAR player adapts the rate at which the server sends frames to the rate at which frames are actually displayed at the client. Since this feedback control is monitoring end-to-end performance, it is able to adapt without actually determining the cause or location of the resource bottleneck. This feedback loop runs continually, and thus can detect changes in resources, such as loss of effective network bandwidth due to increased traffic on a network segment.

A second advantage of dropping frames at the server rather than letting them be lost randomly in the pipeline is that the server can select which frames to drop intelligently, taking MPEG frame dependencies into account. Restricting the number of frames sent also makes it more likely that all packets needed to reconstitute a frame will arrive, rather than partial sets of packets for several frames, which would result in none of those frames being displayable. We see that lowering attempted resource consumption can actually improve quality, because the resources that are used contribute more effectively to what is delivered to the user. This part of the player also shows how we use user-specified quality—frame rate—as a guide rather than a guarantee when availability of resources cannot be controlled. The player will not deliver more quality than requested, but in cases of limited resources, will scale back from the requested quality.

A simple way to utilize the second variable quality dimension is by switching the resolution manually via menu buttons. If the user chooses a lower resolution, the load on the delivery pipeline will drop and the feedback mechanism will increase the frame rate, up to the selected bound. This approach is used in the player, and in this way, the user is able to trade quality in one dimension for quality in another.

The feedback control that monitors effective frame rate can adapt playback to the effective end-to-end network and processing bandwidth. However, the network resource can exhibit other kinds of variation, such as changes in latency (such as might arise from a routing change) and burstiness. Another feedback mechanism monitors these effects and adapts by changing the amount of read-ahead at the server (which will affect the amount of buffering at the client). This

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

frame rate	send pattern
2.5	I----------I----------
5.0	I--P--------I--P--------
10.0	I--P--P--P--I--P--P--P--
15.0	I--PB-P--PB-I--PB-P--PB-
20.0	I-BP-BP-BP-BI-BP-BP-BP-B
30.0	IBBPBBPBBPBBIBBPBBPBBPBB

Figure 4. MPEG frame send patterns

mechanism also deals with drift between client and server clocks, as well as synchronization between media streams.

4.3 Complex content

For composing presentations, two main operations are needed: playing streams in parallel and concatenating streams. In order to play several video streams at the same time we have replicated the entire video pipeline, as shown in Figure 5. In designing this architecture it is important to clearly separate program structures that exist for every stream, and must be replicated, from stream-independent structures. Moreover, a stream-independent notion of time, rather than frame or audio sample numbers, is useful because several streams with potentially different frame or sample rates are supported. Hence, the player measures time in terms of milliseconds from presentation start at normal play speed. The mapping between this logical time and the real time measured by the system clock depends on the actual play speed. Timed events such as displaying of a frame or playing a block of audio samples are managed by an alarm clock module using event list scheduling. The clock blocks the control process until the next event needs to be processed.

The player does not currently support playing several audio streams. In contrast to video with an output window for each stream, multiple audio streams would need to be merged and sent to a single output device.

Concatenating streams is the other basic feature needed for complex presentations. The component streams may be retrieved from different servers with potentially different latencies to the client. Two simple solutions for switching between streams were considered, but deemed unsatisfactory. The first is simply to reinitialize the video pipeline with a connection to the server of the second stream. This procedure, however, takes too long and results in a large gap in the output stream. The second possibility is to replicate the entire pipeline for concatenated streams, too. At the time for the switch, the second pipeline is ready and can start quickly. This approach is excessively wasteful of resources. Because some presentations may concatenate many short streams, many pipelines would be needed, each of them consisting of three processes and two large buffers. Moreover, a way of joining the pipelines to one output window would have to be found. The approach implemented in the player is somewhere between these two alternatives, and avoids extremes in presentation disruption and resource use. For all streams being concatenated, a connection to the server, including the VB process, is maintained during the entire presentation. Having all connections ready allows quick switching between them. The rest of the pipeline is not replicated. All VB processes write to the same B1 buffer, as shown in Figure 6. The second buffer in B1 is needed for prefetching as explained below.

The access of several VBs to B1 needs to be synchronized. The merging of streams is implemented in B1. The buffer accepts data only from the active and the prefetching VB process and ignores (drops) data from inactive ones. This mechanism can be implemented transparently to the VB processes. The timing of the switches between streams is driven by the CTR process. An additional control structure maintains the concatenation-related data and sends the appropriate commands to the respective servers.

The video pipeline has to deal with data from several streams, which may be encoded with different characteristics such as different resolutions and frame rates. One end of the pipeline may still be processing frames of the first stream when the other end has already begun to handle the second stream. Hence, stream-dependent information cannot simply be maintained in global status variables, but is associated with every frame in the pipeline. In this way, every stage can detect a change of parameters and adapt its local state accordingly at the right time.

Concatenation of audio streams has been implemented in the same way. We use a uniform stream abstraction for both video and audio, which allows certain parts of the implementation to be used in both cases.

When playing a presentation with several components, all streams need to be kept synchronized. At the client side, the timing of all streams is based on the system clock, and synchronization is ensured by sharing this time base. It is nec-

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

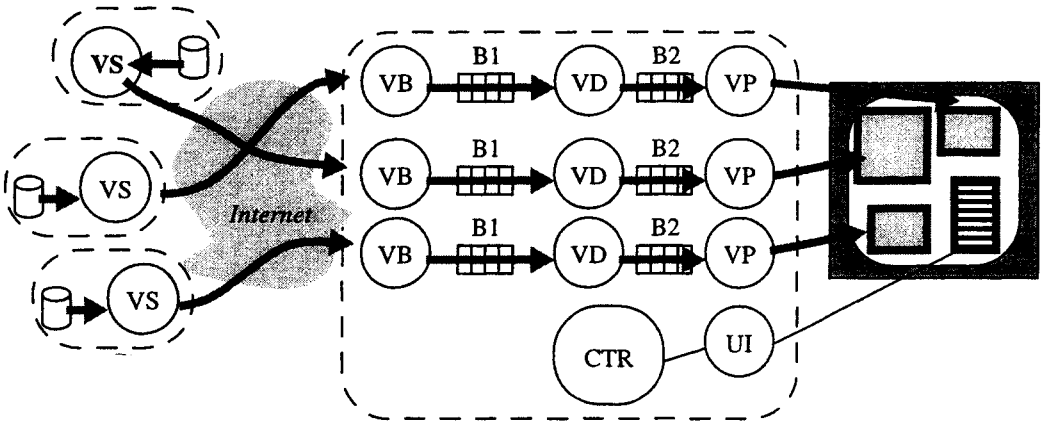


Figure 5. Multiple video streams

essary, however, for the data to be provided by the servers in a timely manner and to arrive at the client when it is needed. To hide network latency and other processing time, the servers work some time ahead of the client. A second software feedback mechanism controls this server workahead time and keeps server clocks synchronized with the client clock. During playback, this mechanism ensures synchronization.

Starting up a stream is trickier in the concatenated-stream architecture than in the single-source case. A command is sent to the server through the control connection. The server then starts sending the data at the specified frame rate. Hence, there is a latency between sending the command and receiving the first frame at the client. Because other parts of a presentation may be being played while this stream is starting up, the client cannot stop its timers and wait for the first frame. If the latency were known, the client could send a command the appropriate amount of time before the scheduled start time. However, obtaining a reliable latency estimate by pinging the server, for example, is difficult because in a congested network the variations in latency are too large. This effect is due in part to the use of a reliable protocol for transmitting the commands – since it is not predictable how often a command packet is resent we cannot separate delays due to retransmission from latency in transit. To address this problem we simply send the start command a constant time ahead of the scheduled start time. This simple solution suffers from two problems: If the constant time is too long, the server is started too early, and frames arrive at the client while the end of the preceding stream is still being played. These prefetched frames of the second stream need to be put into a buffer, hence the two queues in buffer B1, shown in Figure 6. If the prefetch time is early enough, this buffer will overflow. This case can be avoided by choosing the prefetch time no longer than the part of a stream the buffer can hold. If, on the other hand, the prefetch time is too short, the server is started too late and there are no frames of the second stream available when it is scheduled to start. This gap cannot be avoided with the current approach, but it rarely occurs in practice.

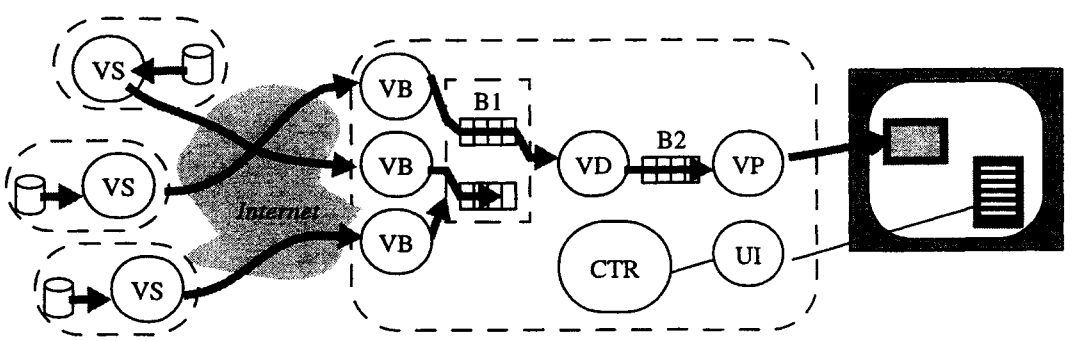


Figure 6. Concatenation

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

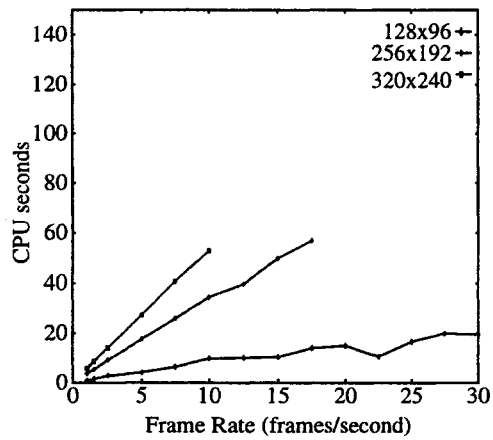


Figure 7. CPU consumption

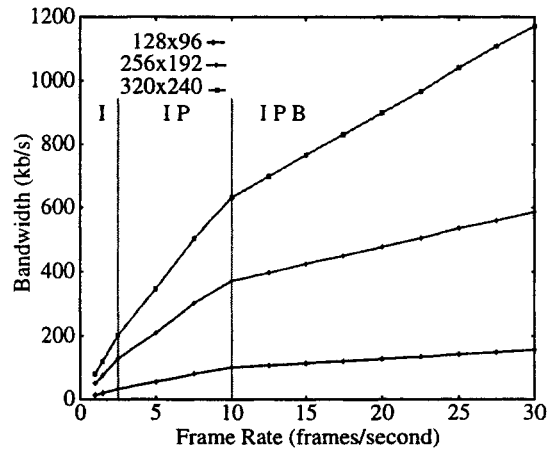


Figure 8. Network bandwidth

5. PERFORMANCE

The experiments discussed below show the effect on resource consumption of varying quality in spatial and temporal resolution dimensions of a video stream. All experiments are done with one Basketball video stored at three different resolutions. Because MPEG bandwidth and decoding times depend on the content of the video, the quantitative results may differ for other clips. The given examples, however, are sufficient to provide some insight into the effects and problems involved.

5.1 Resource consumption

In our environment and player configuration, the client CPU and network bandwidth are usually the scarce resources, and other resource bottlenecks have rarely been observed. Hence, this experiment investigates the effect on client CPU and network bandwidth of varying the quality parameters of spatial resolution and frame rate.

For measuring CPU consumption, it is sufficient to look at the VD process, since all computationally intensive tasks – that is decoding, scaling, and dithering – are done there. A one minute video clip has been run for a set of frame rates and three spatial resolutions. For each run, the CPU time consumed by VD has been queried using the `clock` function of Unix. Figure 7 displays the results.

The figures show that frame rate and resolution can be used to control resource usage trade-offs. With 30% of the CPU for instance, one can get low resolution at 30 frames per second, medium resolution at 6 frames per second, or high resolution at 3 frames per second.

In the same experiment, the average bandwidth for the one minute clips was measured, as shown in Figure 8. We see that with network capacity, resolution and frame rate can be traded for each other. At 300 kb per second, for instance, 4 frames per second can be played at high resolution, 7.5 frames per second at medium resolution, or full frame rate can be obtained at low resolution with only about half of the allotted bandwidth. High bandwidth, however, could not be utilized, because the CPU becomes the bottleneck. Note that the bandwidth graphs consist of three linear parts. These sections are caused by the send pattern of MPEG frames: At 1, 1.5, and 2.5 frames per second only I-frames are sent, at 5, 7.5, and 10 frames per second P-frames are added, and B-frames are included for higher rates.

5.2 Adaptation with two quality dimensions

The feedback mechanism provides automatic fine-grained adaptation of the frame rate. Resolution changes are rather coarse grained and are currently controlled by the user. The graphs in Figures 9 and 10 show how these two mechanisms interact. Each figure represents one presentation. The rate at which frames are sent is shown as well as the rate at which frame are displayed. At the times indicated in the figures, the resolution was changed by the user. The feedback then adapted the frame rate to the new resource requirements.

The presentation in Figure 9 is retrieved from a server on the same machine as the client. In this case, the client CPU becomes the bottleneck. The graphs clearly show that each resolution allows a different frame rate, and that the feedback mechanism quickly adjusts to the new conditions. The feedback mechanism usually adapts in .5 frames-per-second steps. After resolution switches, the adaptation rate is temporarily increased to rapidly find the appropriate frame rate.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

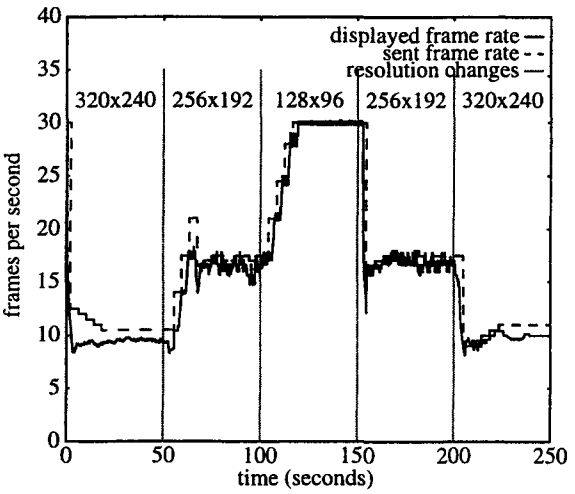


Figure 9. Adaptation, CPU scarce

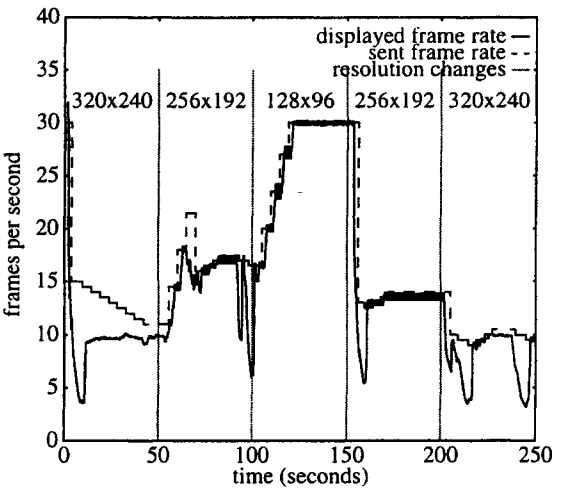


Figure 10. Adaptation, remote server in Germany

The same experiment has been run over the Internet. The remote server was located at the University of Kaiserslautern, Germany, 20 hops away from the client at OGI. On this link, out-of-order delivery of UDP packets can occur, which requires the player to reorder them. Moreover, frames may have to be split into several UDP packets. If one of them is lost in the network, the entire frame must be dropped. This dependency makes larger frames more sensitive to packet loss. I-frames, for instance, are more likely to be lost than B-frames, but the latter cannot be decoded without their reference frames, anyway. Hence, the player performs quite poorly if the network is overloaded, and it depends heavily on adaptation to prevent this situation. The graph in Figure 10 shows an almost ideal performance which was measured on a Saturday morning when there was little load on the network. For a heavily congested network, adaptation may even fail entirely, which shows that the reaction to resolution changes needs to be better integrated with the feedback.

6. CONCLUSIONS

This paper has described the architecture of a distributed multimedia player that integrates support for advanced adaptive QoS control, complex presentations, and real-time transmission of continuous media streams from multiple remote servers. The adaptive QoS control supports dynamic adaptation of video frame rate and resolution, allowing the player to scale its resource consumption. This wide-range adaptability has proven effective in running the player over Internet connections of up to 28 hops and across networks with several orders of magnitude variation in available bandwidth.

Despite this early success, many open research issues remain. One of the most challenging issues is the global coordination of feedback mechanisms across multiple quality dimensions, streams, and players. Regarding coordinated feedback among quality dimensions, the current implementation supports automatic, feedback-based adaptation of frame rate, and manual adaptation of resolution. We are working on the integration of the adaptation mechanisms in both these dimensions to deal with variations in resource availability. However, this integration is non-trivial. Network bandwidth requirements can be calculated from the structure of the MPEG file. The server extracts this data from its frame indices. Similarly, CPU consumption for decoding is approximately linear to resolution and frame rate, while CPU consumption for dithering is proportional to frame rate. Based on this information, and an error model from the user, the system can evaluate if a resolution switch would improve quality. The problem with this approach, however, is the lack of up-to-date information about the availability of resources that are not the current bottleneck. If a resolution switch shifts the bottleneck, this action may degrade rather than improve quality. We are exploring the use of active resource monitors to address this problem.

Instead of coordinating feedback among multiple streams, the current implementation simply controls adaptation one a per-stream basis, i.e., the feedback mechanisms work independently of each other. This behavior can result in a random resource distribution among streams. An error model extended for multiple stream specifications would allow the user to specify the desired weighting of streams. A coordinated adaptation mechanism would then aim to meet this specification. However, dynamic variations in resource availability and resource consumption requirements of streams can make such

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms

adaptation mechanisms highly complex.

Another difficult open research issue concerns the mapping from quality levels to resource consumption levels. A multimedia system that controls several QoS dimensions and uses several resources needs to have some way of relating QoS and resource requirements. An adaptive application needs to find the configuration with the best quality given the resources available. For resource reservation, on the other hand, it is desirable to find the configuration with the least resource requirements for a specified quality. Both problems require a comparison of configurations with respect to the quality they provide as well as the amount of resources they consume. In deciding which delivery plan is the most suitable, the value of the resulting presentation needs to be related to its cost in terms of resource consumption, and perhaps even its dollar cost.

In addition to the research issues outlined above, we are interested in extending the player's authoring environment and the number of quality dimensions it can handle. The player would be more useful if the current restrictions on composing presentations were relaxed. Only simple synchronization and concatenation of streams is currently supported. An arbitrary combination and nesting of these operators would be more desirable. Staehli[11] proposes a normalization algorithm that could be used to transform such a content description to the currently supported form. It would also be useful to be able to include presentations in other presentations, access content descriptions from remote sites, and support synchronized playing and mixing of several audio streams.

7. ACKNOWLEDGEMENTS

This research was partially supported by grants from Intel and Tektronix, and DARPA contracts F19628-95-C-0193, N6601-97-C-8522, and N66001-97-C-8523.

8. REFERENCES

1. Rowe, L. A., Patel, K. D., Smith, B. C., and Liu, K., "MPEG video in software: Representation, transmission, and playback," *In High Speed Networking and Multimedia Computing, IS&T/SPIE, Symposium on Electronic Imaging Science & Technology,* San Jose, CA, February 1994.

2. Clark, D., Shenker, S., and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms", *Proc. SIGCOMM '92*, Baltimore, MD, August 1992.

3. Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D., "RSVP: A new resource reservation protocol," *IEEE Network,* **7**, pp. 8--18, September 1993.

4. B. Braden, et. al., "Resource Reservation Protocol (RSVP) - Version 1 Functional Specification", *Internet Draft*, July 1996.

5. R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," *Internet Request for Comments 1633*, June 1994.

6. I. Stoica, H. Abdel-Wahab, and K. Jeffay, "On the Duality between Resource Reservation and Proportional Share Resource Allocation," *Proceedings, Multimedia Computing and Networking 1997, SPIE Proceedings Series*, San Jose, CA, February 1997.

7. C. Cowan, S. Cen, J. Walpole, and C. Pu, "Adaptive Methods for Distributed Video Presentation", *Computing Surveys Symposium on Multimedia*, **27**(4), pp. 580-583, December 1995.

8. S. Cen, C. Pu, R. Staehli, C. Cowan and J. Walpole, "A Distributed Real-Time MPEG Video Audio Player", *Proceedings Fifth International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV'95)*, Durham, New Hampshire, USA, April 1995.

9. R. Koster, "Design of a Multimedia Player with Advanced QoS Control", *MS thesis*, Oregon Graduate Institute, January 1997.

10. R. Staehli, J. Walpole and D. Maier, "Quality of Service Specification for Multimedia Presentations", *Multimedia Systems*, **3**(5/6), November 1995.

11. R. Staehli, "Quality of Service Specification for Resource Management in Multimedia Systems", *Ph.D. thesis*, Oregon Graduate Institute, January 1996.

12. D. Maier, J. Walpole and R. Staehli, "Storage System Architectures for Continuous Media Data", *Foundations of Data Organization and Algorithms, FODO '93 Proceedings, Lecture Notes in Computer Science,* **730**, Springer-Verlag, Editor David B. Lomet, pp. 1-18, 1993.

13. S. Cen, "A Software Feedback Toolkit and its Application in Adaptive Multimedia Systems," Ph.D. Thesis, Oregon Graduate Institute, September 1997.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms